CS575 Parallel Computing Project#5
Vectorized Array Multiplication and Reduction using SSE

Shangjia Dong

[1]School of Civil and Construction Engineering, Oregon State University

dongs@oregonstate.edu

1.

What machine you ran this on

Response: I ran this on flip.

- Model name: Intel(R) Xeon(R) CPU X5650 2.67GHz
- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- CPU family: 6
- Model: 44
- L1d cache: 32K
- L1i cache: 32K
- L2 cache: 256K
- L3 cache: 12288K

2.

Show the table and graph

Table 1. Non-SSE & SSE Multiplication Performance

Arraysize	Non-SSE Mult	SSE Mult	Speedup
1000	307.6	1805.37	5.869213264
5000	312.83	1947.83	6.226480836
10000	313.44	1944.39	6.203388208
50000	315.51	1959.63	6.210991728
100000	314.16	1824.35	5.807072829
500000	328.42	1659.55	5.053133183
1000000	314.25	1526.94	4.858997613
5000000	309.54	1300.99	4.202978613
10000000	310.13	1297.72	4.184438784
16000000	306.95	1294.31	4.216680241
20000000	323.09	1263.23	3.909839364
32000000	321.37	1345.48	4.186700688

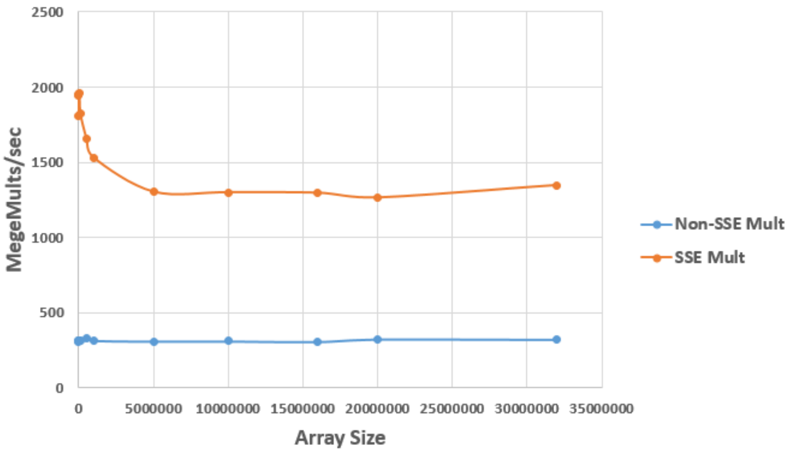

Figure 1. Non-SSE & SSE Array Multiplication Performance

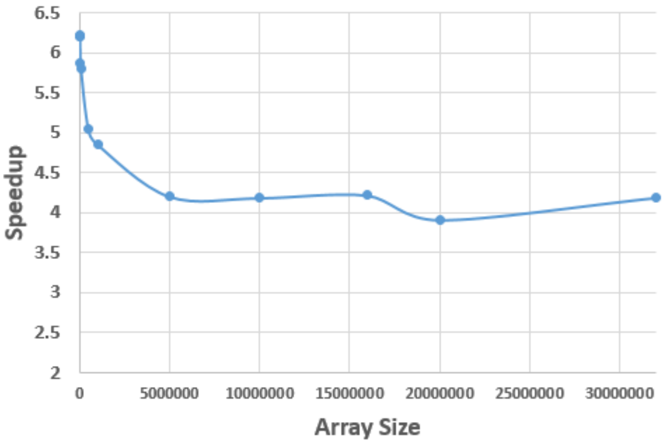

Figure 2. Non-SSE & SSE Array Multiplication Speedup

Table 2. Non-SSE & SSE Multiplication + Reduction Performance

Arraysize	Non-SSE Mult Reduction	SSE Mult Reduction	Speedup
1000	289.11	1766.75	6.110995815
5000	291.7	1907.01	6.537572849
10000	292.08	1926.08	6.59435771
50000	292.32	1937.46	6.627873563
100000	292.37	1939	6.632007388
500000	291.98	1925.12	6.59332831
1000000	291.4	1862.81	6.392621826
5000000	290.92	1812.35	6.229719511
10000000	290.78	1800.78	6.192929362
16000000	291.17	1801.2	6.186076862
20000000	290.91	1765.33	6.068302912
32000000	291.17	1820.5	6.252361164

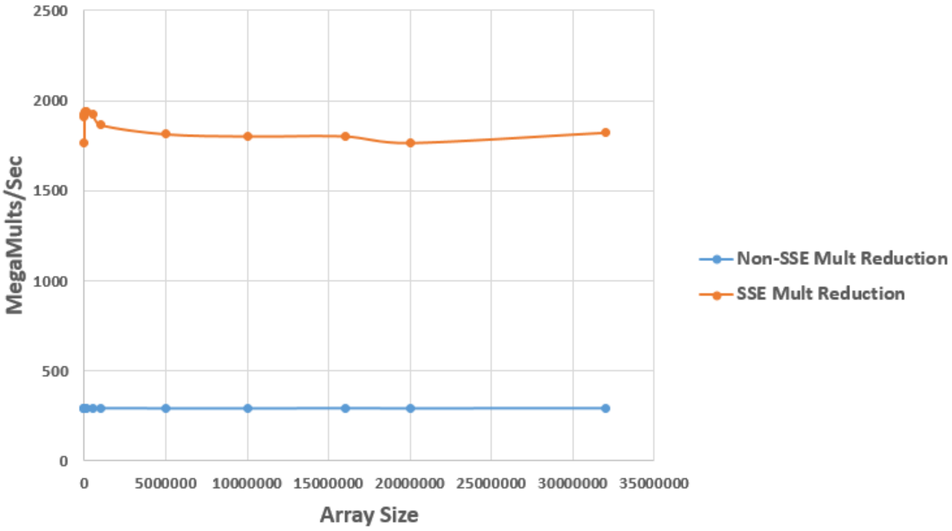

Figure 3. Non-SSE & SSE Array Multiplication + Reduction Performance

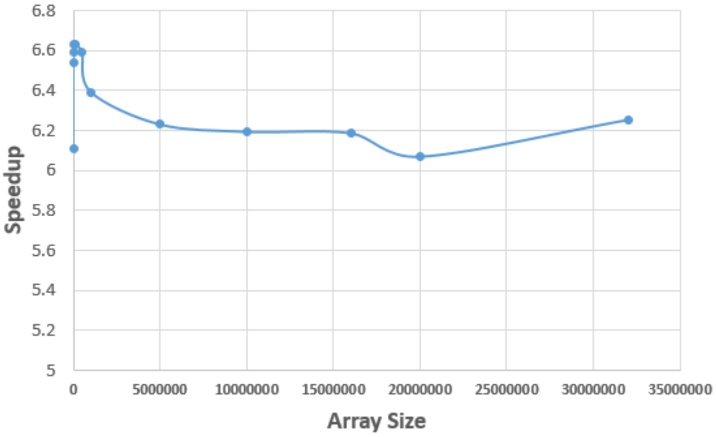

Figure 4. Non-SSE & SSE Array Multiplication + Reduction Speedup

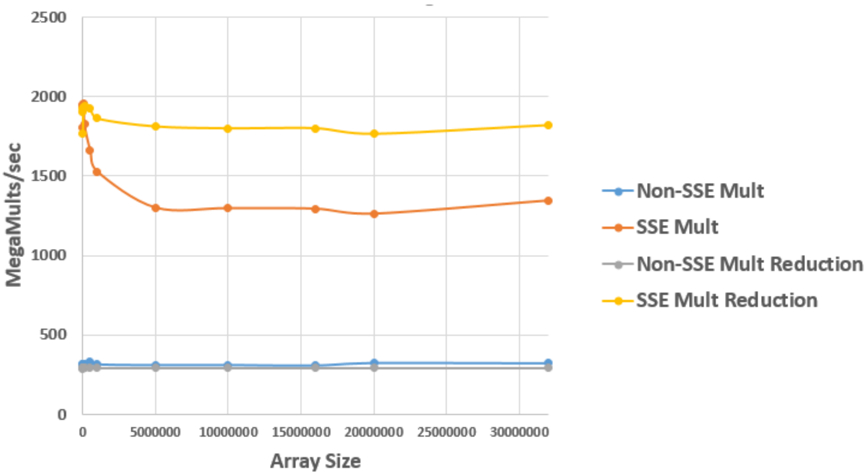

Figure 5. Performance in Summary

3.

What patterns are you seeing in the speedups?

Response: In multiplication only case, when the array size are relatively small, the speedup is pretty high. While the array size increases, the speed up drops a little bit, and tend to be steady after a while.

In multiplication + reduction case, the speed up is low at the beginning, and then increases as the array size becomes bigger, finally it drops to a steady level.

Overall, they all drops at the beginning when the array size increases, and stay steady at a certain point. One thing to notice is, the speedup becomes steady after the array size larger than 5000000.

4.

Are they consistent across a variety of array sizes?

Response: The patterns are similar but the speedup values are not consistent across different array sizes. At the multiplication case, the speedup stays around 4, while in the multiplication + reduction case, the speedup is around 6.

5.

Why or why not, do you think?

Response: The performance of non-SSE multiplication/multiplication + reduction is pretty similar, so the difference comes from the SIMD part.

When we doing the multiplication + reduction, the parameter ”sum” is stored in the memory, this makes it really fast to fetch. Plus, the assembly code is saved on the register, this leads to the faster performance.

Most importantly, the simulation performance changes over time.

6.

Knowing that SSE SIMD is 4-floats-at-a-time, why could you get a speed-up of $<$ 4.0 or $>$ 4.0 in the array multiplication?

Response: The speedup decreases as the array size increases, the value fluctuates around 4, this probably caused by the overhead issue, and also violating the temporal coherence. But the value is above 4 most of the time. This may suggest that the simulation obeyed the temporal coherence reasonably well when the array size larger than 5000000 that we don’t generate a lot of cache misses or we don’t re-load the cache a lot.

7.

Knowing that SSE SIMD is 4-floats-at-a-time, why could you get a speed-up of $<$ 4.0 or $>$ 4.0 in the array multiplication-reduction?

Response: The speedup has a small rise at the beginning, and then dropped a little bit, finally stays relatively steady. The value is above 6 all the time. This might be resulted from the ”sum” function. It is in the memory which is easy to fetch every time. Also, the assembly language are saved on the register, this allows the faster computation as well.